Athens Journal of Sciences- Volume 9, Issue 1, March 2022 – Pages 25-46

Generation, Regeneration and Validation of Binary Secret Keys through Blockchain in IoT Devices

By John M. Medellin[*]

This article operationalizes a mathematical root of trust that can be scaled into protection for Internet of Things (IoT) devices. The initial discussion focuses on gated arrays and the generation of 4-way binary keys. Randomization is used in generation of input and sequence keys giving a unique secret key. The probability of successful attack depends on the number of devices and ordinary implementations are well into one in a billion or more. The paper uses the "epoch" concept; a time-dimensioned interval where more blocks are added to the blockchain. The epochs are selected at random and voting, duration, frequency and key roles are also randomized increasing resiliency. The model does not require constant update of IoT storage; only until such time as communication with others is initiated or a request is received. The substantial savings in processing requirements are significant in IoT. A detailed discussion of the management of the blockchain is provided as well as the necessary blocks enabling the approach. The paper includes a sample dialogue using standard TCP/IP communication structures with security protocols and closing remarks aim at extrapolation to cloud and quantum computing.

Keywords: *blockchain, key management and distribution, internet of things, root of trust, cyber-resiliency*

Introduction

Logic operations, blockchain and key validation/encryption are common terms used in a variety of technologies implemented for protection of computers. This paper proposes a model for interaction of these concepts into an approach that is operationally efficient for devices in the Internet of Things (IoT). These technologies are well known but the ability to have them interact at the right time for protection in this way is novel. The approach specifically lends itself to use in processors that must conserve energy (Huang and Cheng 2002).

This paper is organized into related work, logic gates, blockchain and key exchanges to set preparatory material. Next, the discussion focuses on explaining the randomized election process and key generation, the blockchain components and the interaction of devices along the TCP/IP layers using this model. A simulation-experiment gives an example of the order of magnitude in this approach versus traditional computation-intensive ones. Finally, a brief discussion of extensibility into cloud and quantum computing is provided.

[*]Chief Technical Officer, Medellin Applied Research Concepts, LLC and TruDecision, Inc., USA.

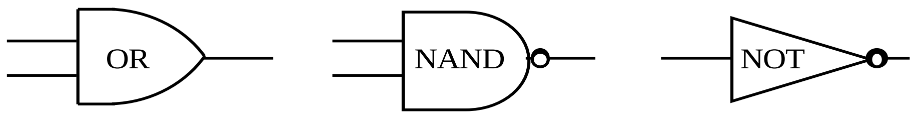
Related Work

Previous work is in three categories; logic gates, blockchain patterns and secret keys.

Logic Operations (Logic Gates)

Logic operations are symbols that operate on two binary input numbers (1 or 0) and yield a third binary outcome. In Figure 1, three operations; OR, NAND and NOT are illustrated in the top part, common symbols for representing them are shown and are used in design for circuits and chips. The matrix below shows the corresponding "truth table" for each "gate" operation above. The format of the truth table shows what happens to the output (Z) when two inputs are entered. There are 4 possible combinations in the two input numbers 1 and 0 with a set of 4 outcomes for the OR and the NAND gates in the example. The NOT X gate only contains 1 input which can have 2 possible values of Z.

Figure 1. *Sample Gate Symbols & Truth Tables*



X	Y	Z
1	1	1
1	0	1
0	1	1
0	0	0

X	Y	Z
1	1	0
1	0	1
0	1	1
0	0	1

X	Z
1	0
0	1

There are 6 operations and outcomes from 2-way binary gates (Ferguson et al. 2010). The truth tables for those gates can be found in Figure 2 (the model that is presented in section "Model Heuristics and Base Operations" requires the usage of 2 way gates, the 1 way gate has been omitted).

Figure 2. *Inputs, Outputs & Gated Outcomes for Different Scenarios*

	Inputs		Logic Gate Outputs (Z)					
Scenario	X	Y	OR	NOR	AND	NAND	XOR	XNOR
1	1	1	1	0	1	0	0	1
2	1	0	1	0	0	1	1	0
3	0	1	1	0	0	1	1	0
4	0	0	0	1	0	1	0	1

A key reason for usage of logic gates is their energy/voltage requirements. Binary result computations based on logic gates will use comparatively low CPU cycles (therefore less energy) versus the requirements of decimal or other base numbering systems. In the case of the division operation, computation of the result can take up to 1 cycle per bit[1] (some typical algorithms like the SHA 384 or higher will contain a payload of 512 hexadecimal values or 512*16=8192 CPU cycles to compute the result). Most algorithms that require key validation today rely on usage of extensive division, multiplication and other operations (Ferguson et al. 2010). By this very virtue, they are more computationally intensive and therefore require more energy to derive. This document assumes and experiments with logic gates to achieve significant resource conservation. Conservation of computational resources is vital to smaller processors (Monk 2017).

Blockchain Design Patterns

Blockchain is a variation of shared data intelligence made famous by Nakamoto (2019); although the author is unknown, it has had a significant impact on creating the concept of shared value exchanges that do not necessarily need to occur through a third-party intermediary. The most famous of these exchange operations is bitcoin. There are literally hundreds of medium exchanges where willing buyers and sellers can contribute value in order to transact between themselves without an intermediary.

This article is not about the usage of such "shared intelligence" to create an exchange for value, rather the usage of the blockchain pattern as a medium for disseminating factual information related to secret keys between participants. Recent academic developments have begun to explore the secrecy and computational advantages of blockchain to communicate in a trust-worthy fashion between members of particular communities (Dinh et al. 2017, Dorri et al. 2017a). Some recent examples of alternate use of blockchain include distribution of sign-on credentials or authentication of agents (Li et al. 2019, Dorri et al. 2017b). These studies rely on exchanging a secret known to the sender and receiver and can be validated by trusted parties who are members of the blockchain (Salman et al. 2018).

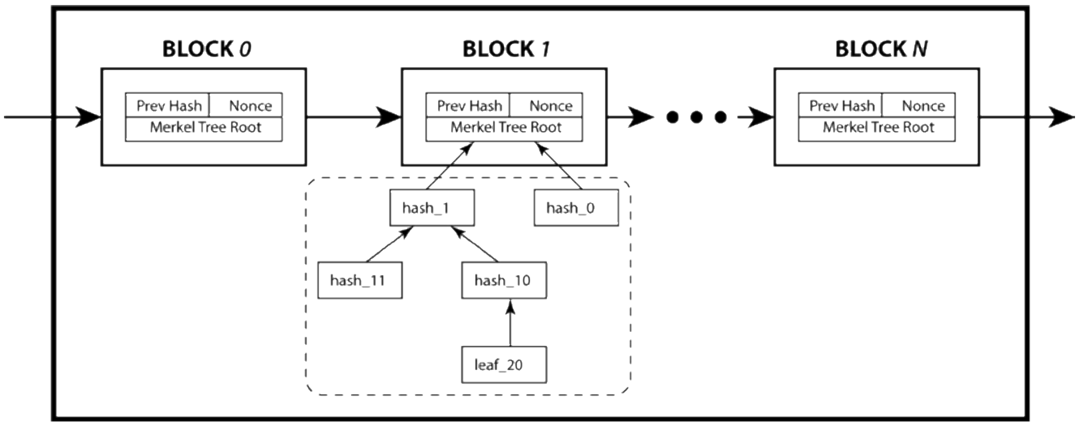
In precursor articles, the author has written about blockchain in the context of usage of this design pattern mostly on the consensus architecture requirements and implementation (Medellin and Thornton 2018). In those studies, comparisons were made to the Byzantine General's Problem; a common shared context problem used to teach the concepts of consensus between participants. This particular technique is complex and very computational but serves as the yardstick to measure efficiency. The focus of this document will branch into measurement of binary operations versus those referenced in higher number system operations to arrive at consensus.

Although there is no authoritative set of components for the blockchain design pattern, there are some typical components. The typical components of the block chain are: the block architecture, a smart contract (which is optional), a consensus model (the ability to validate previous blocks), a set of participants – typically elastic as to volume and a method for encryption using a unique number (the "nonce") which is used once in that encryption (Liang and Wu 2017,

[1]https://projectf.io/posts/division-in-verilog/. [Accessed 17 February 2021]

Christidis and Devetsikiotis 2020, Ongaro and J. Ousterhout 2014, Muralidharan et al. 2018, Ferguson et al. 2010). A partial graphical representation of a typical blockchain set of blocks and attributes is diagrammed and presented in Figure 3.

Figure 3. *Partial Blockchain Sequence and Block*



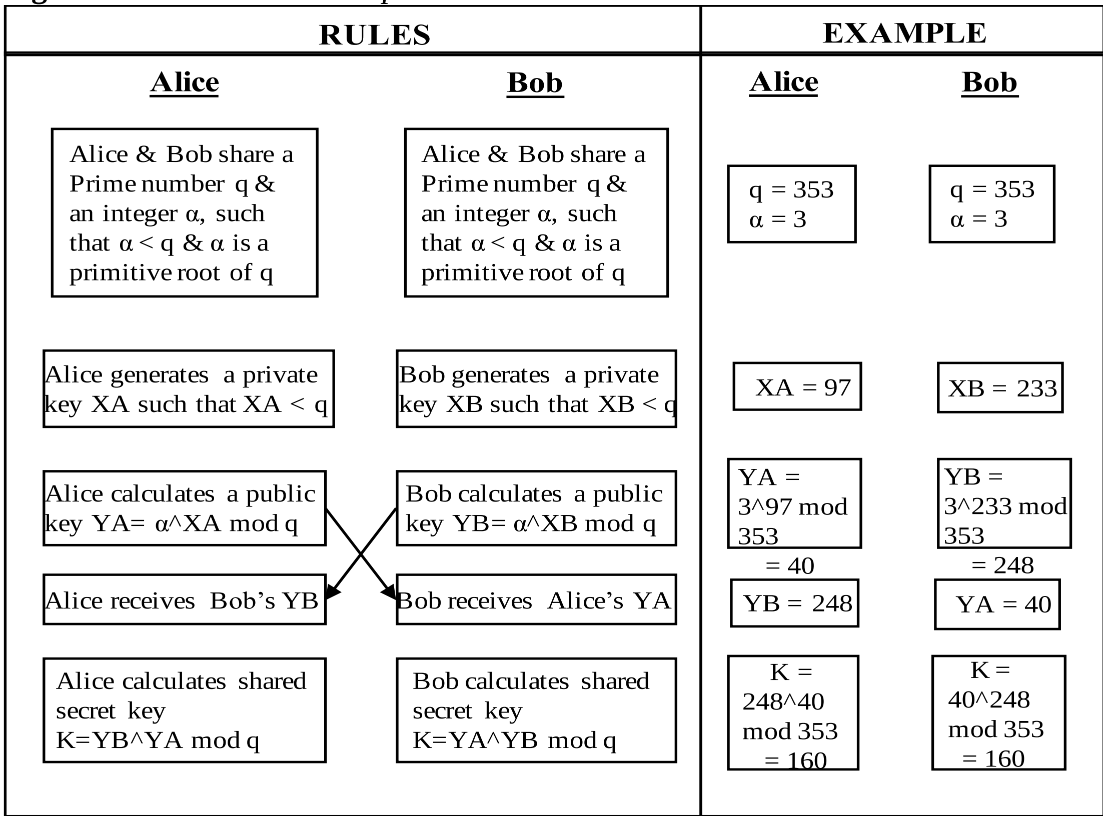
Secret Keys (Trust and Encryption)

Human beings have used the concept of secrets for perhaps millennia. Secrets have been used to preserve and communicate critical information in key situations (Soni and Goodman 2017). These concepts have evolved through the ages and are inherited by computer systems today.

Modern systems create trust between each other by using mathematical formulae that have finite answers. Two computer systems will exchange a particular sequence of numbers and apply a secret formula to determine the veracity of the sequence being received (Johnsonbaugh 2018). If the item checks out then communication can proceed. There are multiple formulae that can be used but the most popular rely on the modulo operations. In modulo operations, a number is divided by another number and the remainder whole-number component is the result. For example, 9 modulo 6 is 3 (9 divided by 6 is 1 with a remainder of 3).

Diffie and Hellman are credited with a widely-used algorithm to validate identity by usage of remainder modulo operations. In this algorithm, actors in send messages to each other encoded with their private keys and arrive at the same number (Kozierok 2017). This is then used to perform encryption on data. The basic DH key exchange is shown below in Figure 4.

Figure 4. *DH Rules and Example*


RULES
EXAMPLE
Alice
Bob
Alice
Bob
Alice & Bob share a Prime number q & an integer α, such that α < q & α is a primitive root of q
Alice & Bob share a Prime number q & an integer α, such that α < q & α is a primitive root of q
q = 353 α = 3
q = 353 α = 3
Alice generates a private key XA such that XA < q
Bob generates a private key XB such that XB < q
XA = 97
XB = 233
Alice calculates a public key YA= α^XA mod q
Bob calculates a public key YB= α^XB mod q
YA = 3^97 mod 353 = 40
YB = 3^233 mod 353 = 248
Alice receives Bob's YB
Bob receives Alice's YA
YB = 248
YA = 40
Alice calculates shared secret key K=YB^YA mod q
Bob calculates shared secret key K=YA^YB mod q
K = 248^40 mod 353 = 160
K = 40^248 mod 353 = 160

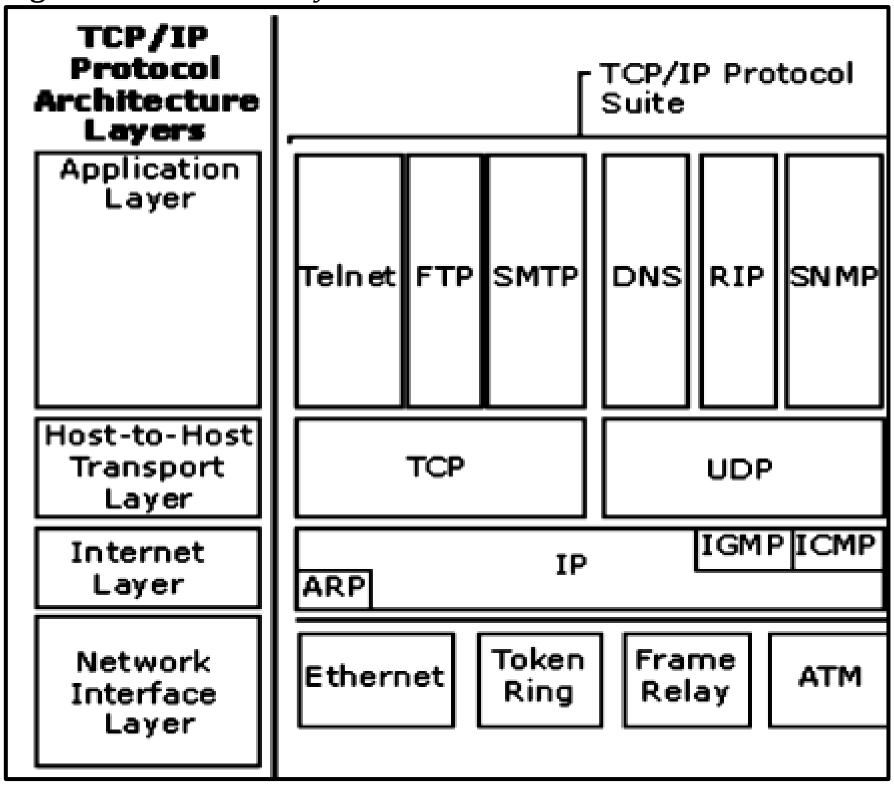
Source: adapted from Stallings (2018).

This article will use a similar approach to conveying trust to other members except that the algorithm to decode the primary message will be regenerated in different binary operations that will be stored along the blockchain's historical blocks.

TCP/IP Concepts

A key assumption of the model in section "Model Heuristics and Base Operations" relies on the usage of TCP/IP as described in Kozierok (2017). The operation of that set of protocols assumes the disaggregation of a message, transmission through physical media and aggregation in the destination. In summary, messages are prepended with routing and lower level information as the data travels down the stack. They are finally transmitted through the physical layer and are de-constructed by each of the layers until they arrive at the application. We are particularly concerned in the handshake that will take place at the with the PPP (point to point) sub-protocol of the ICMP protocol shown in Figure 5.

Figure 5. *The TCP IP Layers and Protocols*



Source: compnetworkgingsec.com.

The PPP dialogue is established at the start of a session and it allows for the usage of CHAP a more modern version of authentication that can be used on the internet layer segment of the protocol. When the two machines are establishing a joint session, the following dialogue occurs:

1. Machine A sends PPP frames to the target address on the network.
2. The receiving Machine B can respond in one of three ways:

   Configure-ACK: parameters accepted, acknowledge and continue.
   Configure-NAK: parameters rejected (and which ones).
   Configure-Reject: ignore.
   Challenge (if challenge met, then ACK).

Section "Model Heuristics and Base Operations" contains a specific example of how the protocol tool set is deployed within the model.

Model Heuristics and Base Operations

In this section, the base matrix, the blockchain components and the base operations are discussed.

The Base Matrix

The first thing proposed in this model is a base matrix that has five columns and contains 4096 rows (more or less data points can be used). Each line contains the following column components:

- The sequence number ("N").
- The private key used in that sequence number ("P").
- The secret key in that sequence number ("Q").
- The gate used in that sequence number ("G").
- The public key in that sequence number ("X").

The base matrix structure is shown in Table 1. The actual matrix values have only binary numbers (0 or 1). The sequence is assumed by location.

Table 1. *Sample Matrix Structure*

P	**Q**	**G**	**X**	**N**
F (0)	T (1)	00(XOR)	T (1)	0
.	.	.	.	.
T (1)	F (0)	01(XNOR)	T (0)	4095

The first line above has the components of P:F(0), Q:T(1), G:00(XOR), X:T(1), N:0. Those values correspond to using the XOR gate on inputs 0,1 and obtaining the number 1. This matrix is never fully implemented in any block on the blockchain rather it is computed by the members each time the members validate each other before beginning exchange of messages.

Blockchain Components

The blockchain components are included in this section.

<u>Blockchain Block</u>

The blockchain block is described in Table 2 and discussed immediately following it.

Table 2. *The Model's Blockchain Block*

Component	**Contents**
Epoch ID	Sequential number for the epoch
Manager secret key and epoch	4096 bit key + original epoch of admission
Public key	4096 bit key generated by manager for the epoch
Gate sequence	4096 x 2 bit key corresponding to the gate being used
Admitted Secret Keys	Sequences of 4096 bits for new members
Deprecated epoch/keys	Deprecated sequences of previous members
Current hash	Previous hash XOR public key XOR gate sequence XOR manager epoch XOR manager secret key XOR current epoch XOR nonce

In Table 2, the epoch id corresponds to the time period of operation (this can be a sequential number or can include the sequential time number and the time clock where the epoch began or variation thereof). The manager key and epoch are the secret key assigned to the manager upon admission and the epoch in which that operation was performed. The public key is the bit sequence of 4096 digits assigned by the manager corresponding to the epoch. The gate sequence is the sequence of 4096 gates corresponding to 2 bits for designating one of 6 binary gates for the epoch being communicated and is optional (if not noted, then the previous epoch's is assumed carried forward). The admitted secret keys are the new members' secret keys for the epoch and must be unique for the epoch, while the next line (deprecated key) corresponds to original epoch and secret key of the members that are being removed. The current hash is a secret number that is known to members to conclude the epoch. All of the above are generated by the manager.

Blockchain Operations

The blockchain operations that are required for this algorithm consist of the consensus method enabled through the manager, the epoch, the election, administrative tasks and conclusion of the epoch. This algorithm relies on epochs and randomization of their duration as explained in Medellin and Thornton (2017). The current manager at random selects a manager for the next epoch, directs when that epoch will begin and produces the next key components of the blockchain. The following paragraphs explain the concepts in greater detail.

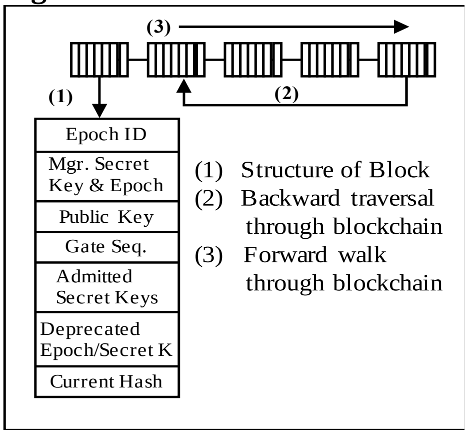
The consensus method is mathematical and is enabled by the random election of a member to perform the duties necessary in the next epoch. A recommended approach is to elect at a minimum one manager and one alternate (which will "wake up" sometime after the manager had to have operated and will assume the duties if one has not done so). Additional alternate managers can be designated in order to increase robustness.

The manager is notified by the previous epoch manager as part of the conclusion of their duties. This previous manager has executed a randomized election by generation of random numbers, partitioning of previous admittances in to a continuous space, assignment of numbers and then assignment to one manager and n-number of alternate managers with instructions to wake up at a random-generated time in the future to execute the next epoch. As mentioned above, the election of a new manager is supervised by the existing manager and is done by announcing the election (for example through individual point to point to all IPv6 addresses or member IDs admitted but not deprecated in the chain). This communication dialogue requests their participation, not all machines need to participate, however those that can must; selection will be from the acknowledging members.

The echo of the machines will be to provide the difference between their secret key total and a random number generated times 4095. The closest (meaning the one with the least numeric difference to that number generated by the manager) and next closest will be assigned as the manager and alternate for the next epoch (if only one alternate is to be used). Only they will be notified of their role and a set of gates will be configured in their IoT arrays to correspond to the logic needed to become a manager as the conclusion of the existing epoch is being executed (for execution of duties in the next epoch). Notification of new parameters will happen in broadcast for those that are in that communication mode, otherwise adjacent machines will be used in the IP protocol to bring themselves current (Kozierok 2017).

As mentioned above, the manager is responsible for execution of the administrative tasks of generating a new Public key and generation of new gate sequences. Typically, both of these will not be done in a single epoch, only occasionally will the gate sequence be targeted for regeneration. Based on that data, the members will re-generate their internal secret key in the base matrix to reflect the current epoch. The model relies heavily on the ability to traverse through the blockchain in order to ensure proper results, this is shown in Figure 6.

Figure 6. *Blockchain Traversal Example*



Admission and deprecation of participants will follow a similar process to what other blockchain algorithms indicate. This will depend on the actual blockchain software to be used (the “fabric”) (Xu et al. 2017). But those duties will also be administered by the manager. After these tasks, the manager will become dormant and the epoch will continue until a new one is declared by new managers or alternates.

Implementation in TCP/IP PPP and TCP/IP CHAP

The preferred method of implementation is by usage of the PPP and CHAP Protocols. PPP initiation begins with first message frames sent containing the user name and password. Once that has been initially validated, the responding machine would send back a challenge using CHAP (challenge/acknowledge). If that challenge was correct then an acknowledgement would occur and the two machines would use private keys to encrypt messages.

Initial Dialogue Between Members

If two members have not previously communicated or have done so in an outdated time frame they must establish trust. The objective of the initial dialogue between two members will be to validate the two parties and establish trusted communication between them. This process is modeled in TCP/IP because it is probably the most utilized of communication protocols in modern computing and because of its inherent robustness in modeling dialogues on the internet (Kozierok 2017). This example dialogue is provided for illustrative purposes, many other approaches are valid as well. The steps are as follows (it assumes no regeneration of the gate sequence, see Figure 7 also):

1. Participant A consults their blockchain store for the current hash. Participant A sends their original epoch of admittance hash XOR encrypted with the current hash (as a user name) and their original admission secret key XOR encrypted with that epoch's hash (as a password) in message #1 through TCP/IP PPP.
2. Participant B XOR decrypts message #1 with the current hash (for a prior epoch hash) and looks for A's admission secret key in that epoch and any possible deprecation since. If it has been deprecated the process stops. B prepends their epoch of admission key to their admission secret key and XOR encrypts with the current hash prepended with the current epoch and sends that as a challenge to A.
3. Participant A receives message #2 and XOR decrypts the value of B's admission epoch and secret key. It then uses that computed secret key as the encryption key. A next XOR encrypts the computed encryption with the current gate sequence and sends to Participant B as reply to the challenge in message #3.
4. Participant B receives message #3 and XOR decrypts with the current gate sequence. If it matches the encryption key sent then an acknowledge is sent and handshaking is over.

Figure 7. *Handshaking Dialogue*

```
                 Participant A                    Participant B

(1) Consult Blockchain
Op: get current hash
Op: encrypt original epoch and secret
 key with current hash and epoch hash
Send message #1 to B------→(2) Rec: original epoch hash / epoch and secret
                                 key encrypted with hash
                              Op: consult B/C for hash
                              Op: decrypt message with hash & get prior epoch
                              Op: get prior hash of epoch, decrypt secret key
                              Op: verify blockchain: no deprecation
                              Op: If ok, encrypt B's epoch of admit + that secret
                                key with epoch and hash
 (3)←-----------------------------Send message #2
Rec: B's admit epoch & secret key
Op: XOR decrypt B's with hash & epoch
Op: Compute implied B's secret key for encryption
Op: XOR B's implied secret key with gate sequence
Send message #3------------→(4) Rec message #3
                              Op: XOR decrypt B's w gate sequence & verifies
                              computation
←--------------------------------Send: ACK (message #4)
```

Guide: Op=Operation, Send=Transmit, Rec=Receive.

Regeneration of Keys

A critical aspect of the model is the ability to regenerate the secret key for each member through instruction to do so from an existing manager. The secret key for members may be regenerated by taking their private key, the gate sequences and the public key. In addition to regeneration of the secret key through public keys, regeneration of the secret key could be done through usage of new gate sequences instead of the public key (input would be the private key, the public key and a new gate sequence for a new secret key).

Another area of expanded regeneration in this model is the ability to stack the gated algorithm and instead of using one set of gates one would use multiple sets of gate sequences in parallel to add further complexity protection. These areas have not been researched at the moment and are expected to be further detailed at a future point in additional research.

Attack Resiliency

A protection scheme's ability to resist intrusion depends on how robust the scheme is and how potentially dangerous such exploits are to the correct functioning of the protected asset (Knapp and Langill 2015). A powerful aspect of the model documented in this article is the ability to increase or decrease the mathematical frequency and payload of the keys used to validate identity. In some cases, the requirement may be for a very high level of protection and in some it will not require as much. This translates into more computational abilities required to fulfill them and therefore more resources (Arnberg et al. Patent Application).

In the two subsections below these implications are discussed by using the base matrix of 4096 rows and 4 columns described above and testing against the probability of a brute-force attack (one in which all possible payloads are used). In the second subsection, additional variations are discussed to further complicate the attack surface.

Attacks on the Previously-Described Base Matrix

Previous segments have described a 4 by 4096 binary base matrix. In order for the attacker to begin an attack, they must have a valid secret key of admission, that epoch's hash, the prior epoch hash and the current epoch's hash. All of these are binary arrays of 4096 rows and the attacker would have to guess these correctly (see Table 3 for attack success probabilities).

Table 3. *Brute Force Attack Success Probabilities*

#	P(x) Secret Key	P(x) Hash Key	P(x) Epoch Hash	P(x) Epoch	P(x) Combined
1	1 / (2^ 4096)	1 / (2^ 4096)	1 / (2^ 4096)	1 / (2^ 4096)	1 / Extremely high #
2	1 / (2^ 2048)	1 / (2^ 2048)	1 / (2^ 2048)	1 / (2^ 2048)	1 / Extremely high #
.⍰⍰⍰	.⍰⍰⍰	.⍰⍰⍰	.⍰⍰⍰	.⍰⍰⍰	.⍰⍰⍰
128	1 / (2^ 32)	1 / (2^ 32)	1 / (2^ 32)	1 / (2^ 32)	1 / (2^ 1,048,576)
256	1 / (2^ 16)	1 / (2^ 16)	1 / (2^ 16)	1 / (2^ 16)	1 / (2^ 65,536)
512	1 / (2^ 8)	1 / (2^ 8)	1 / (2^ 8)	1 / (2^ 8)	1 / (2^ 4,096)
1024	1 / (2^ 4)	1 / (2^ 4)	1 / (2^ 4)	1 / (2^ 4)	1 / (2^ 256)
1536	1 / (2^ 3)	1 / (2^ 3)	1 / (2^ 3)	1 / (2^ 3)	1 / (2^ 81)
2048	1 / (2^ 2)	1 / (2^ 2)	1 / (2^ 2)	1 / (2^ 2)	1 / (2^ 16)

Table 3 depicts the number of machines (#) and the combined probabilities of success in initiating a brute force attack in a static protection scheme if the model was never regenerated through a new gate sequence or new primary keys. In this case, the protection level might be adequate for somewhere around 1,500 to 1,900 IoT machines; the raw probability numbers for the 1,536 members is in one in 242 sextillions which would be an extremely high level of protection in most cases. In addition, however, a key aspect of this paper's contribution is the ability to vary and regenerate parameters based on epochs which will add further complexity and protection versus attacks; that will be discussed next.

The Dynamic Nature of the Model

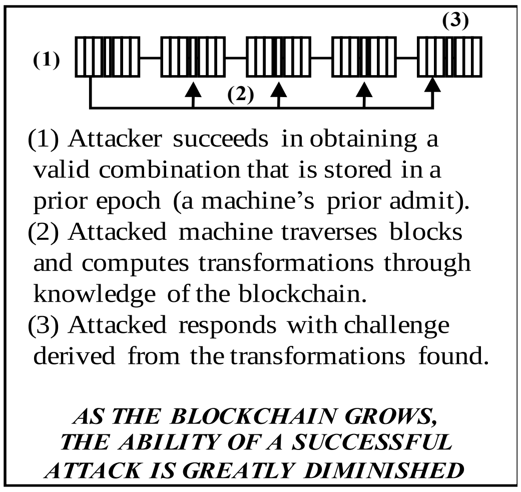
Upon admission of a new member into the network, the manager will generate a private key consisting of a binary value list (for example, 4096 as per the description above) and will compute the value of the member's secret key by taking the generated private key with the epoch's public key as inputs and using the existing gate sequence. The private key and the blockchain payload thus far will be communicated to the newly admitted member. The secret key will be published in the epoch's blocks.

One of the duties of a manager is to publish a new public key in each epoch (a new set of gates can also be published). Each member has the responsibility of regenerating their current secret key by using their private key and the new public key as inputs to the gate sequence. This current secret key is crucial in the hand-shaking algorithm. As discussed above, the frequency of epochs is a random variable selected by the current manager; the manager at random (within tolerances of the number of members and the volume/speed of the network) will announce when the next epoch will begin to the next manager/alternate(s).

The regeneration of the secret keys during every epoch by every member creates another layer of computational complexity for an attacker. Similar to the added complexity afforded by the SHA algorithms this incremental iterative requirement is significant since not only must the attacker know the current parameters but be able to traverse the blockchain in order to continue the dialogue with the intended machine (Stallings 2018). Even if the brute force attack is successful in generating sequence of relevant keys the attacker must have knowledge of the blockchain in order to respond to the challenge. The traversal of blocks is very necessary in order to be able to respond, an incorrect response is ignored and the machine under attack simply does not continue.

As more epochs occur and more blocks are added to the blockchain the added complexity for lack of knowledge will be overwhelming on the attacker. The attacker will need to have intimate knowledge of prior epochs going back to the attacked machine's admission in order to respond (including if there are deprecations of that machine or if gate sequences have been regenerated). The distance of these transformations will increase in similar fashion as that described in the SHA protocol transformations (at one point will exceed the number of transformations in SHA by the additional epochs beyond the transformations in the particular SHA version being used) (Stallings 2018). A diagram of this distancing process can be seen in Figure 8, if gate change or deprecation events are introduced they will require either evaluation or regeneration of private keys further adding complexity for the attacker

Figure 8. *Increased Complexity Through More Blockchain Blocks*



(1) Attacker succeeds in obtaining a valid combination that is stored in a prior epoch (a machine's prior admit).
(2) Attacked machine traverses blocks and computes transformations through knowledge of the blockchain.
(3) Attacked responds with challenge derived from the transformations found.

AS THE BLOCKCHAIN GROWS, THE ABILITY OF A SUCCESSFUL ATTACK IS GREATLY DIMINISHED

Resource Consumption Evaluation

This section presents a simulation experiment on the model.

Formal Requirements

The formal requirements are included in the appendix and heavily rely on the **Z** language (pronounced z-ēd) and is included in the appendix.

Experimental Model Construction

The experimental model focuses on simulating a very simple Modbus/TCP IoT Operational Technology (OT) network as described in Bartelt (2011) and includes the following messages:

- Device status (90 messages).
- Correction sequence (13 messages).
- Correction feedback acknowledge (13 messages).
- Additional 116 messages distributed at random (twice the amount of the previous 3) over the 90-day period.

The simulation is executed in quarter year intervals for a 5-year period, and calculates the cpu cycles required for different size key exchanges. One is for a 512-bit sequence (a smaller subset of the 4096 previously described), another is for the 4096-bit sequence and a third is for 512 bytes. The third is introduced to compare to SHA 384 or higher order SHA protocols.

The simulation for the model described in this document assumes the load of 16 epochs with evenly distributed machines until 96, 192 and 384 machines are admitted into the blockchain eco-system for the bit-based keys. The 512- decimal simulation admits 96, 192 and 384 in one load operation for each instantiation. Keys are regenerated every quarter year, 2% of the machines are admitted and deprecated every period and there are up to 116*2=232 key exchanges per machine per period. D-RAFT elections are not included in the simulation since they only impact one machine (except for the acknowledgement from the participant machines).

Estimation of Non-Volatile Storage Requirements

The usage of blockchain requires the provision of non-volatile storage where the blocks will reside. A critical assumption of the blockchain model is the ability to replicate the data in all the devices that are participants in the network. This characteristic requires estimation of the storage requirements for each (a function of the number of devices, the structure of the blockchain blocks and the number/ types of blocks that will be added to the blockchain as operations occur). The model requires that admission keys be stored, new public keys and gate sequences, deprecation of keys and finally the items which are required for management (the manager key/admission epoch, the epoch ID and the current hash). Table 4 identifies the storage requirements for the 96, 192 and 384-member machine networks. It estimates the initial load and then estimates the addition/ deprecation of 2 devices a quarter for every 96 devices. The summary lines at the bottom identify the non-volatile storage requirements for the blockchain at the quarter, year and 5 year marks only for the bit-based keys (the totals are rounded up to ensure the blocks will be written).

Table 4. *Base Load & Operating Store Non-volatile Storage Requirements per Device*

Base Load Store Component		**Blockchain**		**Base Load**					
Item	Assumption	512 bit	4096 bit	512 bit	4096 bit	512 bit	4096 bit	512 bit	4096 bit
# Devices >>>	16 Epochs			96	96	192	192	384	384
Epoch	key length	512	4096	8,192	65,536	8,192	65,536	8,192	65,536
Mgr. Key+Epoch	32 + key bits	544	4128	8,704	66,048	8,704	66,048	8,704	66,048
Primary Key	key length	512	4096	8,192	65,536	8,192	65,536	8,192	65,536
Gate Sequence	key length	512	4096	0	0	0	0	0	0
Admit Secret Key	key length	512	4096	49,152	393,216	98,304	786,432	196,608	1,572,864
Deprecated Key	32 + key bits	544	4128	0	0	0	0	0	0
Current Hash	key length	512	4096	8,192	65,536	8,192	65,536	8,192	65,536
Total bits				82,432	655,872	131,584	1,049,088	229,888	1,835,520
Total bytes				10,304	81,984	16,448	131,136	28,736	229,440
Expressed in KB				10.3	82	16.5	131.2	28.7	229.4
Operating Store Component		512 bit	4096 bit	**Regeneration, Admission & Deprecation @ 2 mach./qtr.**					
Time Period >>>				Quarter	Quarter	Year	Year	5 Year	5 Year
Epoch	32 bits	32	32	32	32	128	128	640	640
Mgr. Key+Epoch	32 + key bits	544	4128	544	4,128	2,176	16,512	10,880	82,560
Primary Key	key length	512	4096	512	4,096	2,048	16,384	10,240	81,920
Gate Sequence	key length	512	4096	0	0	0	0	0	0
Admit Secret Keys	key length	512	4096	1,024	8,192	4,096	32,768	20,480	163,840
Deprecated Keys	32 + key bits	544	4128	1,088	8,256	4,352	33,024	21,760	165,120
Current Hash	key length	512	4096	512	4,096	2,048	16,384	10,240	81,920
Total bits				3,712	28,800	14,848	115,200	74,240	576,000
Total bytes				464	3,600	1,856	14,400	9,280	72,000
Expressed in KB				0.5	3.6	1.9	14.4	9.3	72
Storage per machine for # machines on network				Q/512	Q/4096	Y/512	Y/4096	5Y/512	5Y/4096
96 Devices (KB)				10.8	85.6	12.2	96.4	19.6	154.0
192 Devices (KB)				19.1	151.2	26.8	211.4	68.0	532.2
384 Devices (KB)				35.5	282.3	55.9	441.2	164.7	1288.3

Estimating CPU-Cycle Requirements

The estimated CPU-cycle requirements for three different scenarios (512 binary array, 4096 binary array and SHA 384+) are given in Table 5 (a description of the assumptions follows).

Table 5. *Estimated CPU-Cycle Loads Under Various Keys*

	512 bits	**4096 bits**	**SHA 384+**
Message #1			
Receive	1	1	1
Decode	32	256	0
Fetch Block	2	2	0
Validate/Derive	16	128	512
Format Challenge	16	128	512
Message #3			
Receive	1	1	1
Decode	32	256	0
Validate/Derive	16	128	512
Acknowledge	1	1	1
Total CPU Cycles	117	901	1,539

Table 5 identifies the number of cycles required for each operation under the dialogue mentioned in Figure 7. The assumptions made for each are as follows:

- Receive: one cycle to receive the message (store in buffer).
- Decode: for the bit-based keys it is two messages (user name and password) using a 32-bit chip, bitwise operations (e.g., (512/32) * 2 = 32 CPU-cycles).
- Fetch block (from the block chain): one for fetch and one for the receive.
- Validate/Derive: bit-wise operations similar to the ones in decode for the bit keys, 512 -digit division for the SHA key.
- Format Challenge: same as previous operation.
- Acknowledge: one cycle to send standard "ACK" message.

Results Discussion

The results may be found in the appendix are for a low interaction system (in practice the interactions in process control may be much higher). In addition, the frequency of key negotiation will depend on the ability to isolate the processes from potential attack and the necessity to regenerate encryption keys. Those considerations will need to be evaluated by the designer of the system in addition to the specific component of the blockchain itself. This document has provided one example of the gated component but the variations to the payloads in the model are very large.

An important concept illustrated above is the radical difference in cpu cycles depending on the algorithm used for generation and regeneration of keys between IoT devices. Some of these may be able to devote a high degree of cycles as for example in the ARM Cortex-MO Processor which can yield a 0.87 MIPS (million instructions per second) at a speed of 2.25MHz and is a three-stage cycle processor[2]. Given a typical 20% "headroom" (additional processing unused) it can deliver around 0.232 million full instruction capacity and a simple key negotiation would not begin to scratch the surface. However, more cycles would be required if the SHA negotiation were something more resilient such as prime number keys (something that is utilized in higher safety systems for example).

In addition, however, there are other processors that have considerably less power such as those mentioned in Lallement et al. (2017) which may still be industrially viable but with much less cpu power (e.g., 7Hz) to devote to protection. These processors do exist in implementations and need more care in determining which protection algorithm to use so they do not spend most of their effort in processing large keys.

[2]https://static.docs.arm.com/ddi0432/c/DDI0432C_cortex_m0_r0p0_trm.pdf?_ga=2.84689169.908795371.1542781838-925179195.1542781838. [Accessed 17 March 2021]

Potential in Other Technologies

Section "Resource Consumption Evaluation" assumed that the IoT processors will physically process the instructions required. In addition, it has assumed that the regeneration of the keys is evenly distributed (at least through the experiment) and can count on a static space where adversaries can try to hack into the system. This section discusses potential variations in cloud computing and the ability to provide some resilience in quantum processor attacks through randomization.

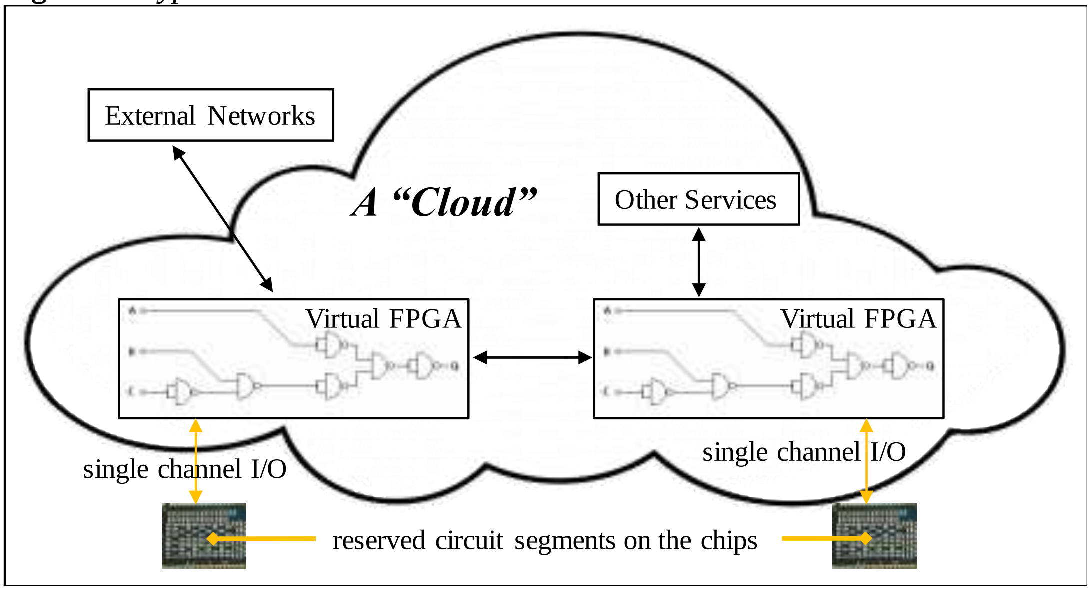
Cloud Computing

Cloud technology affords great flexibility and efficiency in managing computing resources. What used to cost millions and take years to build can now be achieved in fractional time and cost. Cloud computing can help in this model by providing logic-gated arrays in virtual clouds.

These "virtual gated arrays" (sometimes also called "FPGA" or Field Programmable Gated Arrays) can be designed with gated sequences in mind and can also be scaled to address the needs of a particular project with little effort or time. Virtual FPGAs as they are called can now be sourced from many of the public clouds and instead of having to purchase the hardware, one can now design in that environment and deploy very quickly and inexpensively.

Figure 9 illustrates a potential implementation of the algorithm in Virtual FPGAs, taking the load off the IoT processors themselves by managing all communication (and protection) in the cloud while delivering payloads in a secure, isolated channel. Several offerings exist to both house the processors in a virtual environment and also the blockchain operations in public clouds (such as Amazon Web Services).

Figure 9. *Hypothetical Architecture in a Public Cloud*



Quantum Computing

Quantum computing is a promising technology enabling massive processing in fractional time. Peter Shor introduced an algorithm that was able to overcome the finding of such primes considered later useful in deciphering the SHA style cryptography. This was later confirmed by Spiller in finding such prime numbers using Shor's algorithm. While not commercially viable at this point, in the future these designs could potentially find answers to primary key exchanges and cryptography fractions of what it takes today.

Prime numbers are finite & finding them can be quite complex. Several algorithms exist to confirm the existence of primality (Xu et al. 2017, Nakamoto 2019). It is conceivable that a quantum computer could break the secrets of the model presented in this article. However, one must also consider that the algorithms can be regenerated at random intervals adding infinity to the puzzle. Others such as those documented in sub-subsection "Regeneration of Keys" can be used to randomize and also confuse the attacker requiring them to initiate again (and on, and on….).

One additional word on the above, the regeneration of keys consumes additional resources and one must not be careless to fall into regeneration in intervals that are very frequent because that is wasteful. Rather, one needs to design the system with an analysis of the attacker strength and provide for sufficient regeneration in order to defeat it. In the end however, if a sufficiently powerful quantum computer is used, these efforts might not be enough.

Conclusion

This document presented a method for interaction between logic gates, blockchain and key generation that has some definite savings in computation for the IoT. This is an initial discussion on a new approach to key generation and regeneration. Risk areas still exist in this method and they are being explored as this document is being submitted for consideration. The author believes in adequate protection based on the asset values and potential for real intrusion. This document advocates for a different approach that can increase or decrease computational complexity (and resources) depending on the protection objectives.

References

Arnberg A, Van Ermel Scherer R, Medellin J (n.d.) *Device for implementing ubiquitous connectivity and protection software for IoT devices*. US Patent Application 62/371, 003.

Bartelt T (2011) Industrial automated systems; instrumentation and motion control. Clifton Park, New York: Delmar Cengage Learning.

Christidis K, Devetsikiotis M (2020) Blockchains and smart contracts for the internet of things. *IEEE Access* 4(1): 2292–2303.

Dinh TTA, Wang J, Chen G, Liu R, Ooi BC, Tan K-L (2017) *BLOCKBENCH: a framework for analyzing private blockchains*. Retrieved from: https://arxiv.org/pdf/1703.04057.pdf. [Accessed 19 March 2019]

Dorri A, Kanhere SS, Jurdak R (2017a) Towards an optimized blockchain for IoT. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 173–178.

Dorri A, Kanhere SS, Jurdak R, Gauravaram P (2017b) Blockchain for IoT security and privacy: the case study of a smart home. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 618–623.

Ferguson N, Schneier B, Kohno T (2010) *Cryptography engineering, design principles and practical applications*. Indianapolis, Indiana: Wiley Publishing, Inc.

Huang S-Y, Cheng K-T (2002) *Formal equivalence checking and design debugging*. Norwell, Massachusetts: Kluwer Academic Publishers.

Johnsonbaugh R (2018) *Discrete mathematics*. 8th Edition. New York: Pearson Education, Inc.

Knapp E, Langill J (2015) Industrial network security, securing critical infrastructure networks for smart grid, SCADA and other industrial control systems. 2nd Edition. Waltham, Massachusetts: Syngress Elsevier.

Kozierok C (2017) *The TCP/IP guide, a comprehensive, illustrated internet protocols reference*. San Francisco, California: No Starch Press, Inc.

Lallement G, Abouzeid F, Cochet M, Daveau J-M, Roche P, Autran J-L, et al. (2017) *A 2.7pJ/cycle 16MHz with 4.3nW power-off ARM Cortex-M0+ core in 28nm FD-SOI*. Leuven, Belgium: ESSCIRC, hal-01788172.

Li D, Du R, Fu Y, Au MH (2019) Meta-key: a secure data-sharing protocol under blockchain-based decentralized storage architecture. *IEEE Networking Letters* 1(1): 30–33.

Liang X, Wu T (2017) Exploration and practice of inter-bank application based on blockchain. In *The 12th International Conference on Computer Science & Education (ICCSE 2017)*, 219–224.

Medellin J, Thornton M (2017) Simulating resource consumption in three blockchain consensus algorithms. In *"MSV '17" International Conference on Modeling, Simulation & Visualization Methods*, 21–27.

Medellin J, Thornton M (2018) Performance characteristics of two blockchain consensus algorithms in a VMWare hypervisor. In *International Conference on Grid & Cloud Computing and Applications "GCA '18"*, 10–17.

Monk S (2017) *Programming FPGAs, getting started with Verilog*. New York: McGgraw Hill.

Muralidharan S, Murthy C, Nguyen B, et al. (2018) Hyperledger fabric: a distributed operating system for permissioned blockchains. Retrieved from: https://arxiv.org/pdf/1801.10228.pdf. [Accessed 19 March 2019]

Nakamoto S (2019) *Bitcoin: a peer-to-peer electronic cash system*. Retrieved from: www.bitcoin.org. [Accessed 19 March 2019]

Ongaro D, Ousterhout J (2014) In search of an understandable consensus algorithm. In *Proceedings ATC'14 USENIX Annual Technical Conference USENIX*, 305–319.

Salman T, Zolanvari M, Erbad A, Jain R, Samaka M (2018) Security services using blockchains: a state of the art survey. *IEEE Communications Surveys & Tutorials* 21(1): 858–880.

Soni J, Goodman R (2017) *A mind at play, how Claude Shannon invented the information age*. New York: Simon & Schuster.

Spivey J (1988) *Understanding Z, A Specification Language and its Formal Semantics*. Cambridge, UK: Cambridge University Press.

Stallings W (2018) *Cryptography and network security, principles and practice*. 7th Edition. London, United Kingdom: Pearson Education Limited.

Xu X, Weber I, Staples M, Zhu L, Bosch J, Bass L, et al. (2017) A taxonomy of blockchain-based systems for architecture design. In *2017 IEEE International Conference on Software Architecture*, 243–252.

Appendix

A1. Formal Requirements Definition

This section outlines the basic formal requirements for implementation of the blockchain model. The schemas and operations are enumerated and then the key ones are formally described in **Z**.

A1.1 Schemas and Operations

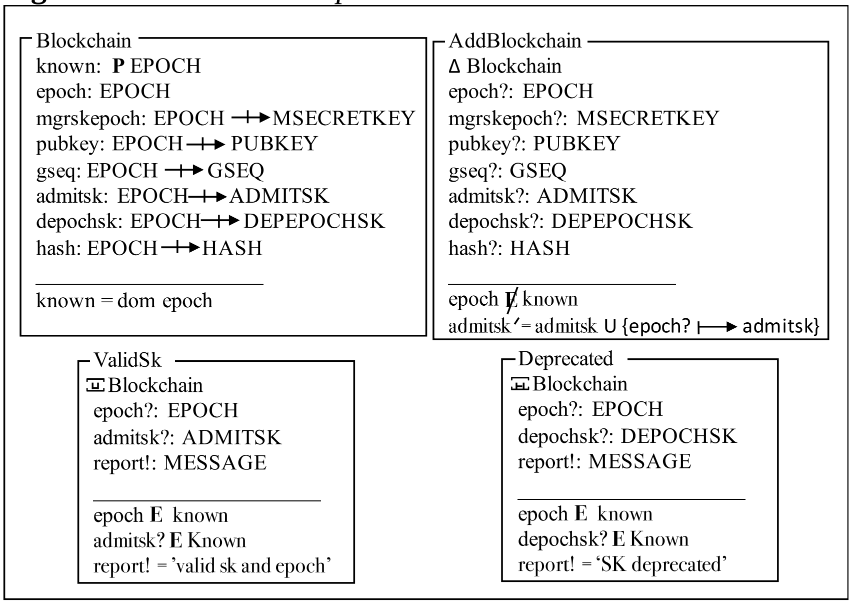
The specification is based on schemas (collections of data primitives) and change operations (dynamic effects) on those schemas. The schemas and operations for the blockchain and member segments of the system are as follows, a checkmark appears next to each if they will be used in the evaluation:

- Schema: Blockchain √ & Operation: AddBlockchain √
- Schema: Message1 & Operation: Message1CreateSend
- Operation: Message1Validate √
- Schema: Message2 & Operation: Message2CreateSend √
- Schema: Message3 & Operation: Message3CreateSend
- Schema: Message4 & Operation: ACK4CreateSend √

A1.2 Sample Schemas and Operations in Detail

The following Z language sample definitions are from the complete specification (it is voluminous and will be published in the future). The **Z** language guarantees the correctness of the specification by mathematical proofs and only those artifacts are translated into actual code (Spivey 1988).

Figure 10. *Schemas and Operations in* **Z**



```
Blockchain
known: P EPOCH
epoch: EPOCH
mgrskepoch: EPOCH ⇸ MSECRETKEY
pubkey: EPOCH ⇸ PUBKEY
gseq: EPOCH ⇸ GSEQ
admitsk: EPOCH ⇸ ADMITSK
depochsk: EPOCH ⇸ DEPEPOCHSK
hash: EPOCH ⇸ HASH
______________________
known = dom epoch

AddBlockchain
Δ Blockchain
epoch?: EPOCH
mgrskepoch?: MSECRETKEY
pubkey?: PUBKEY
gseq?: GSEQ
admitsk?: ADMITSK
depochsk?: DEPEPOCHSK
hash?: HASH
______________________
epoch ∉ known
admitsk′= admitsk ∪ {epoch? ↦ admitsk}

ValidSk
Ξ Blockchain
epoch?: EPOCH
admitsk?: ADMITSK
report!: MESSAGE
______________________
epoch E known
admitsk? E Known
report! = 'valid sk and epoch'

Deprecated
Ξ Blockchain
epoch?: EPOCH
depochsk?: DEPOCHSK
report!: MESSAGE
______________________
epoch E known
depochsk? E Known
report! = 'SK deprecated'
```

A2. Simulation and Results

A structured simulation was constructed in the c language using the gcc compiler in Linux using the -o option. The simulation was run for 20 quarters using the method described. Three key lengths were used (512 & 4096 bit binary and SHA 384+, assumed at 512 byte), the results are in Table 6.

Table 6. *Simulation Results; Key Negotiation for a Single Member*

```
  KEY NEGOTIATION AND CPU CYCLE RESULTS FOR 20 QUARTERS
---------------------------------------------------------------------------------------------------------
Quarter 1  Keys Negotiated = 220  512bit= 25740  4096bit= 198220  sha= 338580
Quarter 2  Keys Negotiated = 170  512bit= 19890  4096bit= 153170  sha= 261630
Quarter 3  Keys Negotiated = 198  512bit= 23166  4096bit= 178398  sha= 304722
Quarter 4  Keys Negotiated = 187  512bit= 21879  4096bit= 168487  sha= 287793
---------------------------------------------------------------------------------------------------------
Year 1   Keys Negotiated 775  512bit= 90675  4096bit= 698275  sha= 1192725
---------------------------------------------------------------------------------------------------------
Quarter 5  Keys Negotiated = 119  512bit= 13923  4096bit= 107219  sha= 183141
Quarter 6  Keys Negotiated = 169  512bit= 19773  4096bit= 152269  sha= 260091
Quarter 7  Keys Negotiated = 129  512bit= 15093  4096bit= 116229  sha= 198531
Quarter 8  Keys Negotiated = 231  512bit= 27027  4096bit= 208131  sha= 355509
---------------------------------------------------------------------------------------------------------
Year 2   Keys Negotiated 648  512bit= 75816  4096bit= 583848  sha= 997272
---------------------------------------------------------------------------------------------------------
Quarter 9  Keys Negotiated = 204  512bit= 23868  4096bit= 183804  sha= 313956
Quarter 10  Keys Negotiated = 158  512bit= 18486  4096bit= 142358  sha= 243162
Quarter 11  Keys Negotiated = 169  512bit= 19773  4096bit= 152269  sha= 260091
Quarter 12  Keys Negotiated = 146  512bit= 17082  4096bit= 131546  sha= 224694
---------------------------------------------------------------------------------------------------------
Year 3   Keys Negotiated 677  512bit= 79209  4096bit= 609977  sha= 1041903
---------------------------------------------------------------------------------------------------------
Quarter 13  Keys Negotiated = 117  512bit= 13689  4096bit= 105417  sha= 180063
Quarter 14  Keys Negotiated = 143  512bit= 16731  4096bit= 128843  sha= 220077
Quarter 15  Keys Negotiated = 120  512bit= 14040  4096bit= 108120  sha= 184680
Quarter 16  Keys Negotiated = 172  512bit= 20124  4096bit= 154972  sha= 264708
---------------------------------------------------------------------------------------------------------
Year 4   Keys Negotiated 552  512bit= 64584  4096bit= 497352  sha= 849528
---------------------------------------------------------------------------------------------------------
Quarter 17  Keys Negotiated = 192  512bit= 22464  4096bit= 172992  sha= 295488
Quarter 18  Keys Negotiated = 222  512bit= 25974  4096bit= 200022  sha= 341658
Quarter 19  Keys Negotiated = 169  512bit= 19773  4096bit= 152269  sha= 260091
Quarter 20  Keys Negotiated = 153  512bit= 17901  4096bit= 137853  sha= 235467
---------------------------------------------------------------------------------------------------------
Year 5   Keys Negotiated 736  512bit= 86112  4096bit= 663136  sha= 1132704
---------------------------------------------------------------------------------------------------------
Year 5 Cum.  Keys Negotiated 3388  512bit= 396396  4096bit= 3052588  sha= 5214132
=============================================================
```

The results in Table 6 are for a single member. This simulation is for a very low requirement in a particular IoT network (for example low risk refrigerated warehouse with zones that vary minimally or for a set of refrigerators in long term storage that do not require daily operation).